Unix / Linux Fundamentals

Slides By: Ynon Perek

ynon@ynonperek.com
http://ynonperek.com

Course Goals

- At the end of the course, you'll
  - Perform common unix tasks from the terminal
  - Create, edit and modify files
  - Use and combine existing unix tools
  - Customise your unix environment

Course Resources

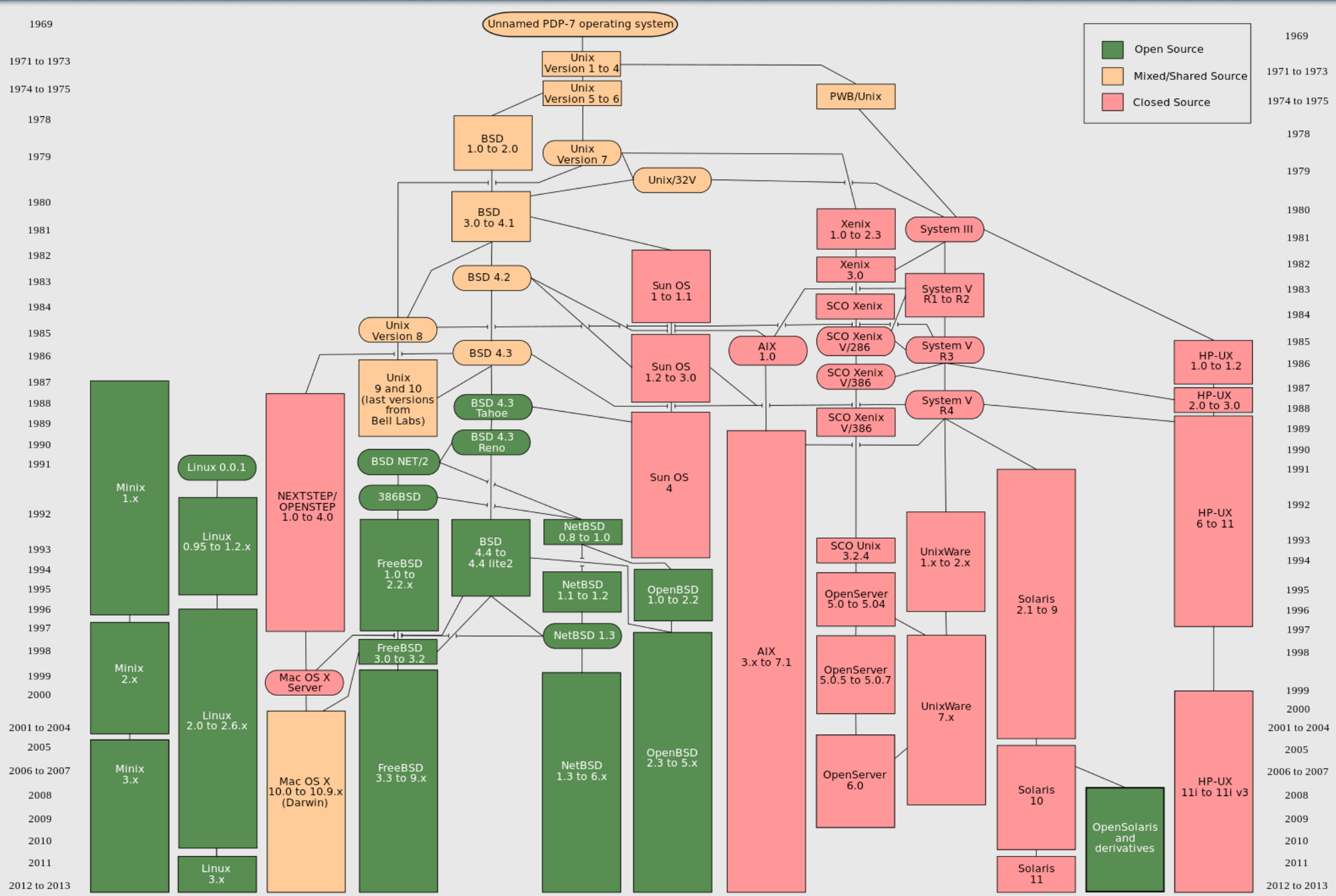
- Slides:
  http://ynonperek.com/static/unix1.pdf

- Labs:
  https://gist.github.com/ynonp/8518427

- Extra Reading:
  http://ynonperek.com/course/unix/about.html

- Unix Tutorials
  http://www.ee.surrey.ac.uk/Teaching/Unix/

How It All Started



Unix Philosophy

- Programs do one thing well
- Programs collaborate for larger goals
- Prefer text formats over binaries
- Documentation is bundled in

A Unix Man Page

```
CAT(1)                          User Commands                          CAT(1)
```

NAME

```
       cat - concatenate files and print on the standard output
```

SYNOPSIS

```
       cat [OPTION] [FILE]...
```

DESCRIPTION

```
       Concatenate FILE(s), or standard input, to standard output.

       -A, --show-all
              equivalent to -vET

       -b, --number-nonblank
              number nonblank output lines

       -e     equivalent to -vE
```

Free Software Movement

- Stallman started GNU project in 1984
- Goal: Bring back the cooperative spirit
- Focus on tools and environment. No full O/S.

Free Software

- Freedom to run the program, for any purpose
- Freedom to study how the program works, and modify it to fit your needs
- Freedom to redistribute copies
- Freedom to redistribute copies of your modified versions

Linux Fit In

- (1991) Free home-brew unix flavour
- Based on Minix and uses GNU

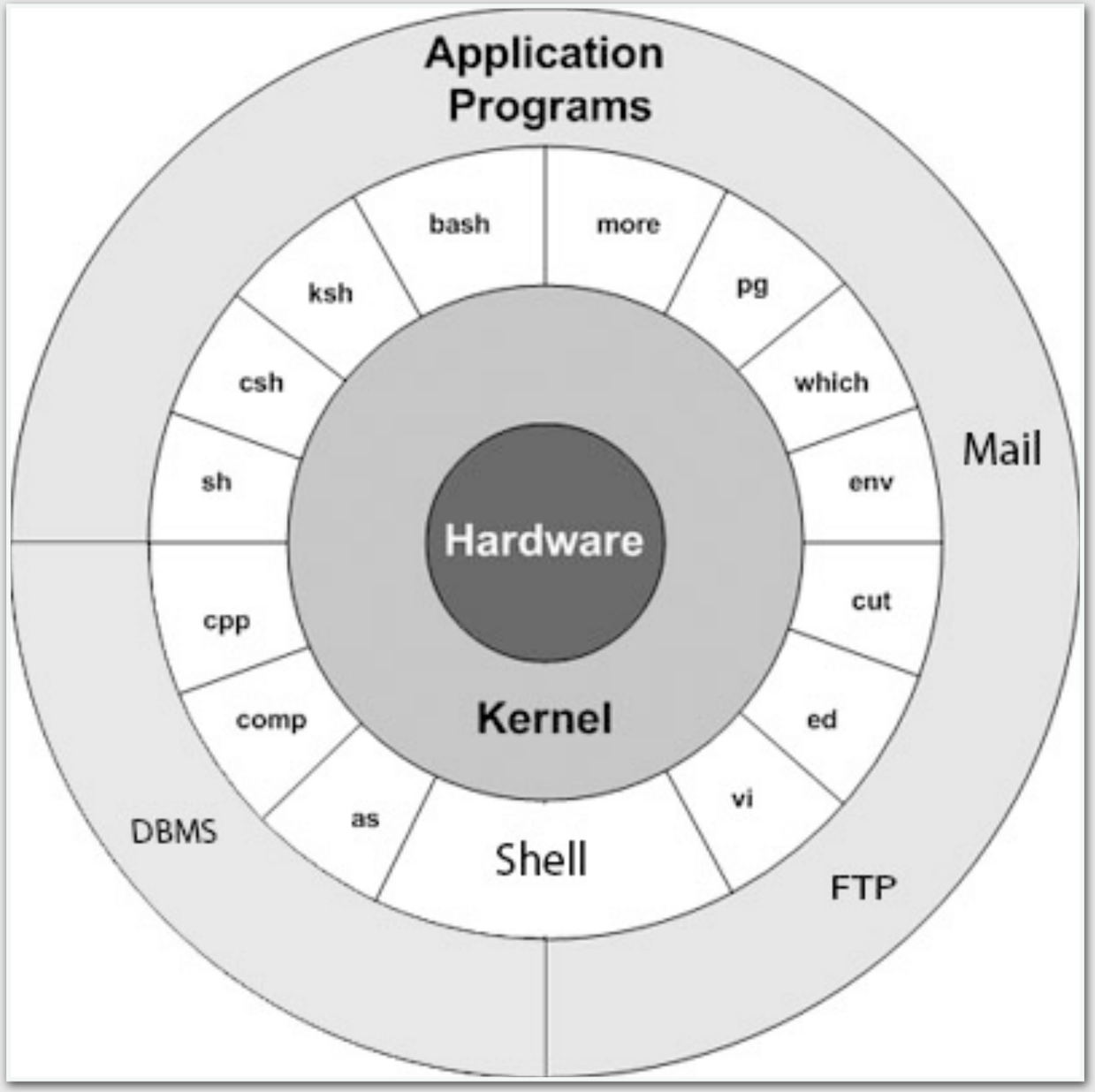
GNU/Linux Today

- Used by small, medium and large companies
- Many different distributions
- Uses free and open source software

Unix Shells In Use

Q & A

The Unix OS

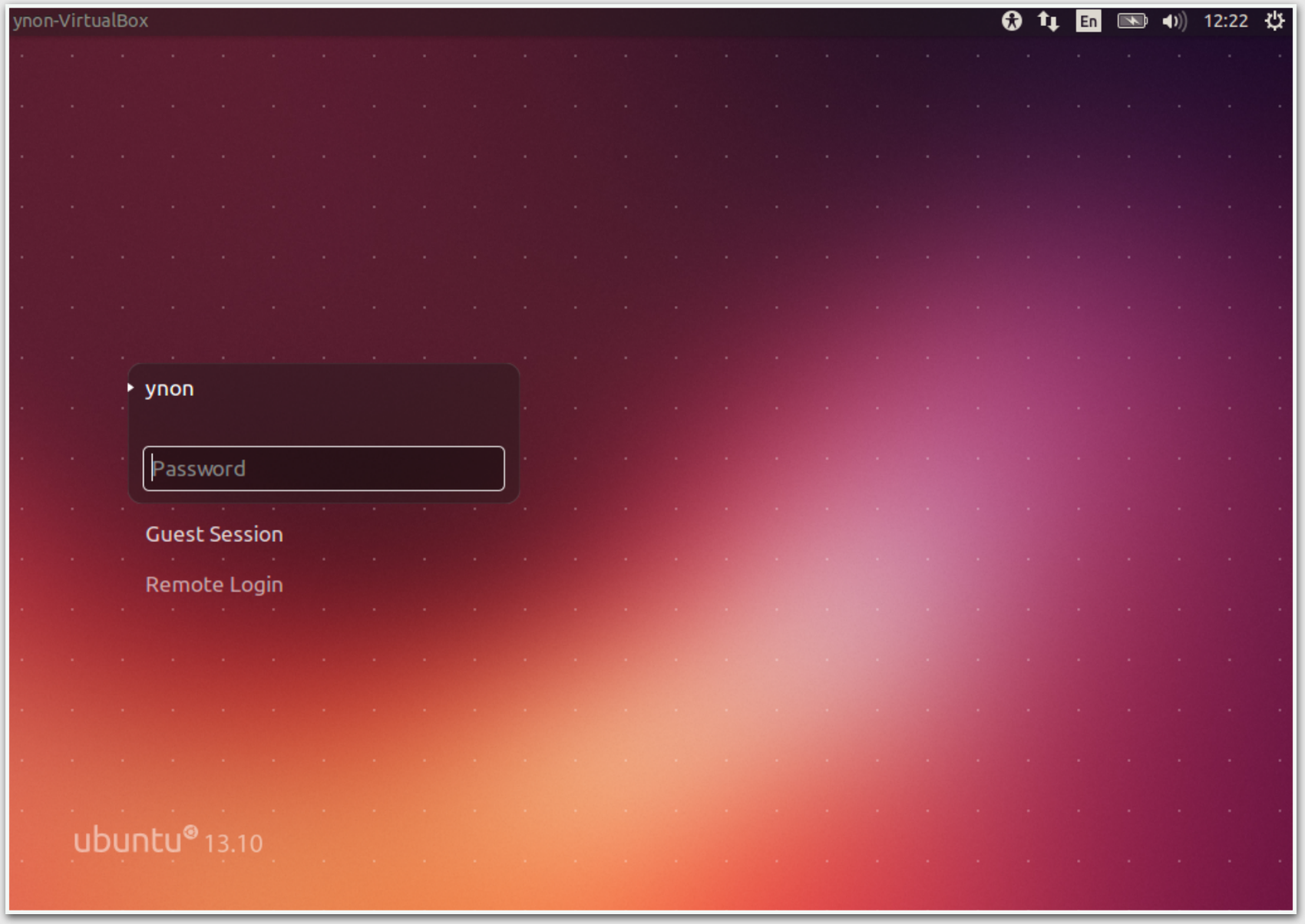

Application Programs
bash
more
ksh
pg
csh
which
sh
env
Mail
Hardware
cpp
cut
comp
Kernel
ed
vi
as
DBMS
Shell
FTP

Logging In

ynonperek — ynon@li453-66: ~ — ssh — 79×23

```
Linux li453-66 3.9.3-x86_64-linode33 #1 SMP Mon May 20 10:22:57 EDT 2013 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Dec 15 11:55:14 2013 from 93-173-152-99.bb.netvision.net.il
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
ynon@li453-66:~$
```

Logging In


ynon-VirtualBox
En
12:22
ynon
Password
Guest Session
Remote Login
ubuntu® 13.10

Hello Shell

```
% echo "Hello Unix"
Hello Unix

% echo $0
-bash

% date
Sun Dec 15 12:41:24 IST 2013

% cal
   December 2013
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

- Basic unix commands and responses
- Don’t type the %. It’s just the prompt marker

Hello Shell

```
% id

uid=501(ynonperek) gid=20(staff)
groups=20(staff),501(access_bpf),12(everyone),61(localaccounts),
79(_appserverusr),80(admin),81(_appserveradm),98(_lpadmin),33(_a
ppstore),100(_lpoperator),204(_developer),398(com.apple.access_s
creensharing),399(com.apple.access_ssh)

% whoami

ynonperek

% groups

staff access_bpf everyone localaccounts _appserverusr admin
_appserveradm _lpadmin _appstore _lpoperator _developer
com.apple.access_screensharing com.apple.access_ssh
```

Unix Shells

- A shell is a program that runs other programs
- Available shells:
  - sh, ash, bash, dash, ksh, zsh
  - csh, tcsh
  - fish, psh, pysh, scsh, wish, zoidberg

How Shell reads you

```
ls -l --color /tmp /home
```

command

options

arguments

How CSH reads you

- command is the name of the program to run
  - executable file
  - internal shell command

How CSH reads you

- options are prefixed with dashes
  - single dash mean single-letter option
  - double dash mean long option
- many commands support both

How CSH reads you

- Some options take arguments:
  - `date -d “01/01/2012”`
  - `cowsay -f dragon hello`

How CSH reads you

- arguments start after the last option

Shell Quoting

- shells read lines word-by-word, and uses spaces to separate words
- Can group words with quotes

```
# send two arguments to
# ls (/home/ynon AND perek)
% ls /home/ynon perek

# send one argument to ls
% ls "/home/ynon perek"
```

Shell Quoting

- Can also use:

```
% ls /home/ynon\ perek

% ls '/home/ynon perek'
```

More Shell Commands

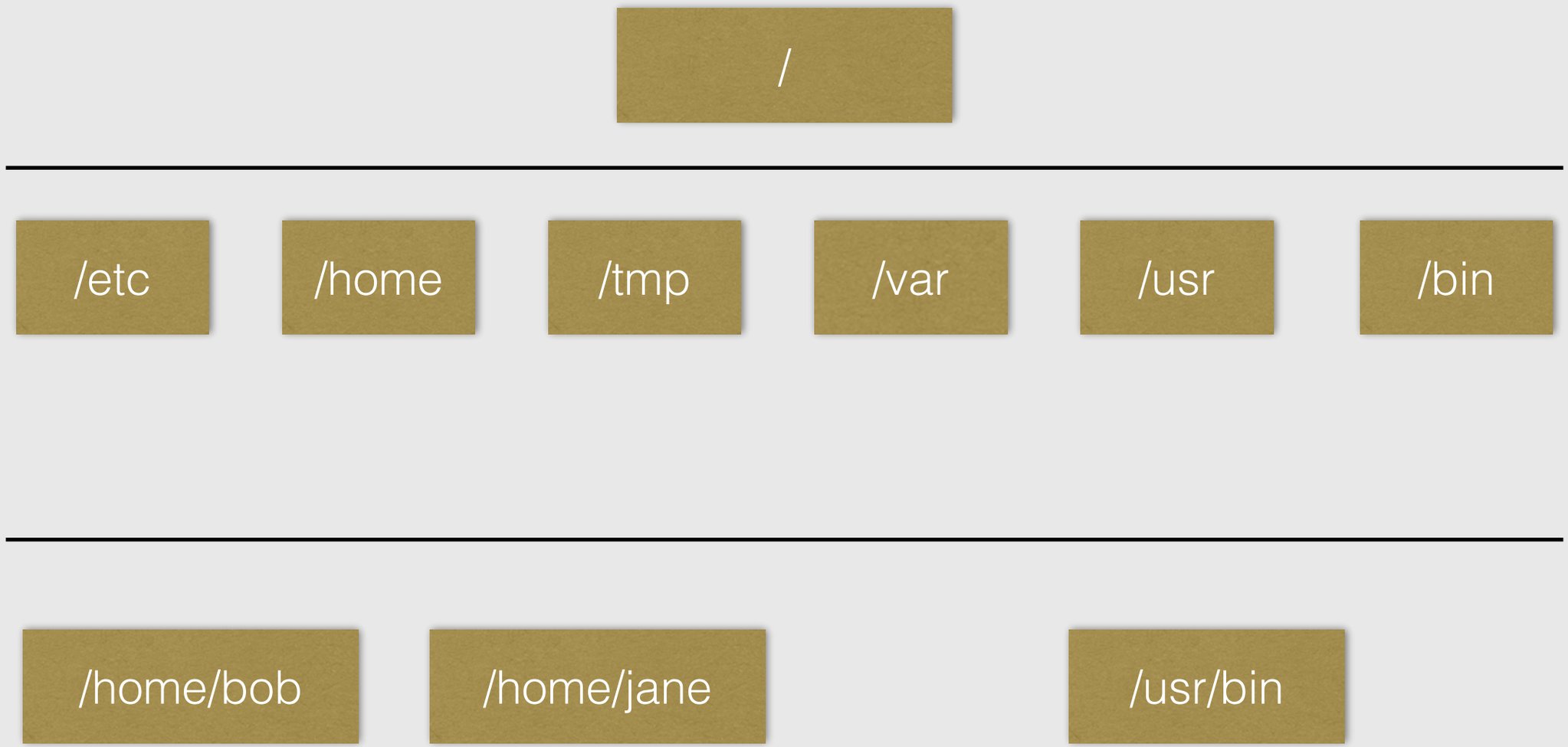
```
# cat shows contents of a file
cat /etc/passwd

# more provides a basic pager
# (use q to quit)
more /etc/passwd

# less has a better pager
# (use q to quit, or h for help)
less /etc/passwd
```

Q & A

Lab: Welcome To Unix

Unix File System

Directory Tree



Directory Tree

- Configuration files are placed in /etc
- For example:
  - /etc/passwd holds all users info
  - /etc/shells holds a list of all installed shells

Directory Tree

- Home directories are located in /home
  - Each user gets her own home
  - Personal documents, photos, bookmarks
  - Program definitions

Directory Tree

- Temporary files are stored in /tmp

Directory Tree

- /var is a good place to anything that's of variable size
  - Server logs
  - Web apps
  - Caches
  - Print spools

Directory Tree

- /usr is where your distribution's extra programs are installed
  - /usr/bin for binaries
  - /usr/lib for library files

Directory Tree

- /usr/local is where you install programs you compiled yourself
  - /usr/local/bin for binaries
  - /usr/local/lib for libraries

Moving Around

```
# print current working directory
pwd

# go back to home directory
cd

# go to /etc
cd /etc

# Now we're in /etc, we can step into /etc/network
cd network

# Now we're in /etc/network, we use .. to get back
# one folder
cd ..

# And we're back in /etc
```

Moving Around

```
# Home directories are marked with ~
# Goes to your home directory
cd ~

# Goes to joe's home directory
cd ~joe
```

Relative vs. Absolute Paths

- A path starting with / is called absolute
- Not influenced by current working directory
- Examples:
  - cd /etc
  - ls /usr/local

Relative vs. Absolute Paths

- A path without a starting / is called a relative path
- Shell automatically prefix it with current working directory
- Examples (assume you’re in /home/joe):
  - cd documents => lead to /home/joe/documents
  - cd .. => back to /home/joe

File Types

```
Ynons-MacBook-Air:qtest ynonperek% ls -l /dev/ttys000 /dev/disk2 .

br--r-----  1 ynonperek  staff    1,   7 12:19 15 דצמ /dev/disk2
crw--w----  1 ynonperek  tty     16,   0 15:18 15 דצמ /dev/ttys000

.:
total 128
-rw-r--r--  1 ynonperek  staff  47440 20:33 29 אוק Makefile
-rw-r--r--  1 ynonperek  staff    369 20:33 29 אוק main.cpp
-rw-r--r--  1 ynonperek  staff   2684 20:33 29 אוק main.o
lrwxr-xr-x  1 ynonperek  staff     11 15:17 15 דצמ passwd -> /etc/passwd
drwxr-xr-x  3 ynonperek  staff    102 20:33 29 אוק qtest.app
-rw-r--r--  1 ynonperek  staff    288 20:33 29 אוק qtest.pro
```

Hidden Files

- Unix shells treat files starting with . as configuration files
- Most programs won’t list them, unless you explicitly ask

Hidden Files

```
% ls -l -a
.
..
.bar
.buz
.foo
Makefile
main.cpp
main.o
passwd
qtest.app
qtest.pro
```

```
% ls -l

Makefile
main.cpp
main.o
passwd
qtest.app
qtest.pro
```

Q & A

Working With Files

```
# Copy a file to a new name
cp /etc/passwd /Users/ynonperek/passwd.new

# Or keep the name
cp /etc/passwd /Users/ynonperek

# Ask before overwriting an existing file
cp -i /etc/passwd /Users/ynonperek

# Overwrite existing files without asking
cp -f /etc/passwd /Users/ynonperek
```

Working With Files

```
# Rename a file
cd /home/ynon
mv image.png flower.png

# Delete a file
rm flower.png
```

Working With Files

```
# Create a new text file
cat > myinfo.txt
name: ynon
email: ynon@ynonperek.com

unix is cool
^D

# Create a new text file using an editor
nedit myinfo.txt
```

Working With Directories

```
# Copy an entire folder
cd /home/ynon
cp -r /usr/local/bin mybin

# Move an entire folder
mv mybin bins_from_local

# Delete an entire folder
rm -r bins_from_local
```

Working With Directories

```
# Crete a new folder
mkdir myapps

# Create a subfolder
mkdir myapps/mytest

# Create a folder with subfolders
mkdir -p game/images/icons

# delete an empty folder
rmdir myapps/mytest
```

Lab: Files and Directories

Wildcard Matching

- Shell was made by lazy people
- Wildcards allow us to type less, by letting the shell do the work
- Every time you use a “special” character, the shell replaces the word containing the special character with a list of file names

Wildcard Matching

- Assume your directory has the following files: game.c, game.h, music.c, music.cpp, music.h, readme.txt, monster.png, hero.png, all.mp3
- Let's examine some wildcards

Wildcard Matching

What You Write	What Shell Reads
*.c	game.c music.c
game.[ch]	game.c game.h
music.?	music.c music.h
a	all.mp3 game.c game.h readme.txt

Wildcard Matching

Character	Meaning
*	Match 0 or more characters
?	Match any single character
[…]	Match 1 character from the ones in brackets

Wildcard Notes

- Wildcards are expanded by the shell (before running any program)
- Use double quotes or backslash to prevent wildcard expansion
- Example:
  - echo “why?”

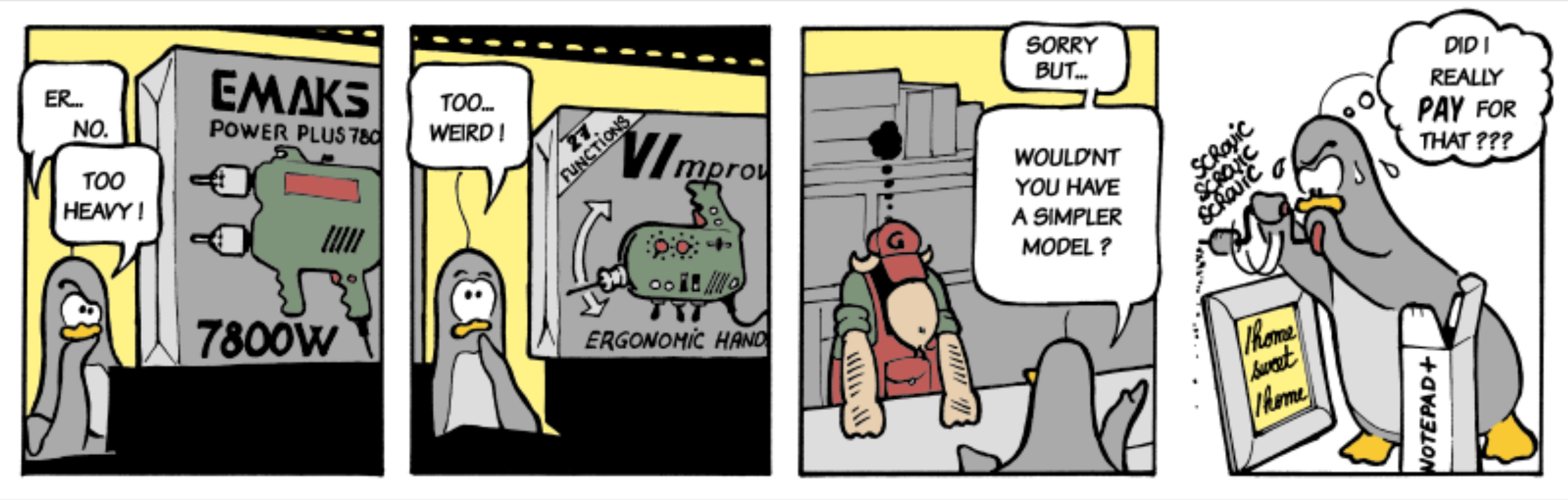
Wildcards and .

- Star wildcard doesn’t catch files starting with a dot
- Use .* if you really need to
- Examples:
  - `ls .*`
  - `cp * .* destination/`

Lab: Wildcards

Powerful Text Editing

Is it worth it?



Vi Story

- (1976) Bill Joy writes vi
- Name vi stands for visual
- Concept: modular text editor
- Part of the Unix spec

Vim Story

- (1991) Developed by Bram Moolenaar
- Intended as a vi port for Amiga
- Free software (charityware)
- Extremely popular

Online Resources

- Best VIM Cheat Sheet:
  http://bencrowder.net/files/vim-fu/

- Derek Wyatt's vim video tutorials:
  http://www.derekwyatt.org/vim/vim-tutorial-videos/

- Vim adventure game:
  http://vim-adventures.com/

- Usevim (vim blog):
  http://usevim.com/

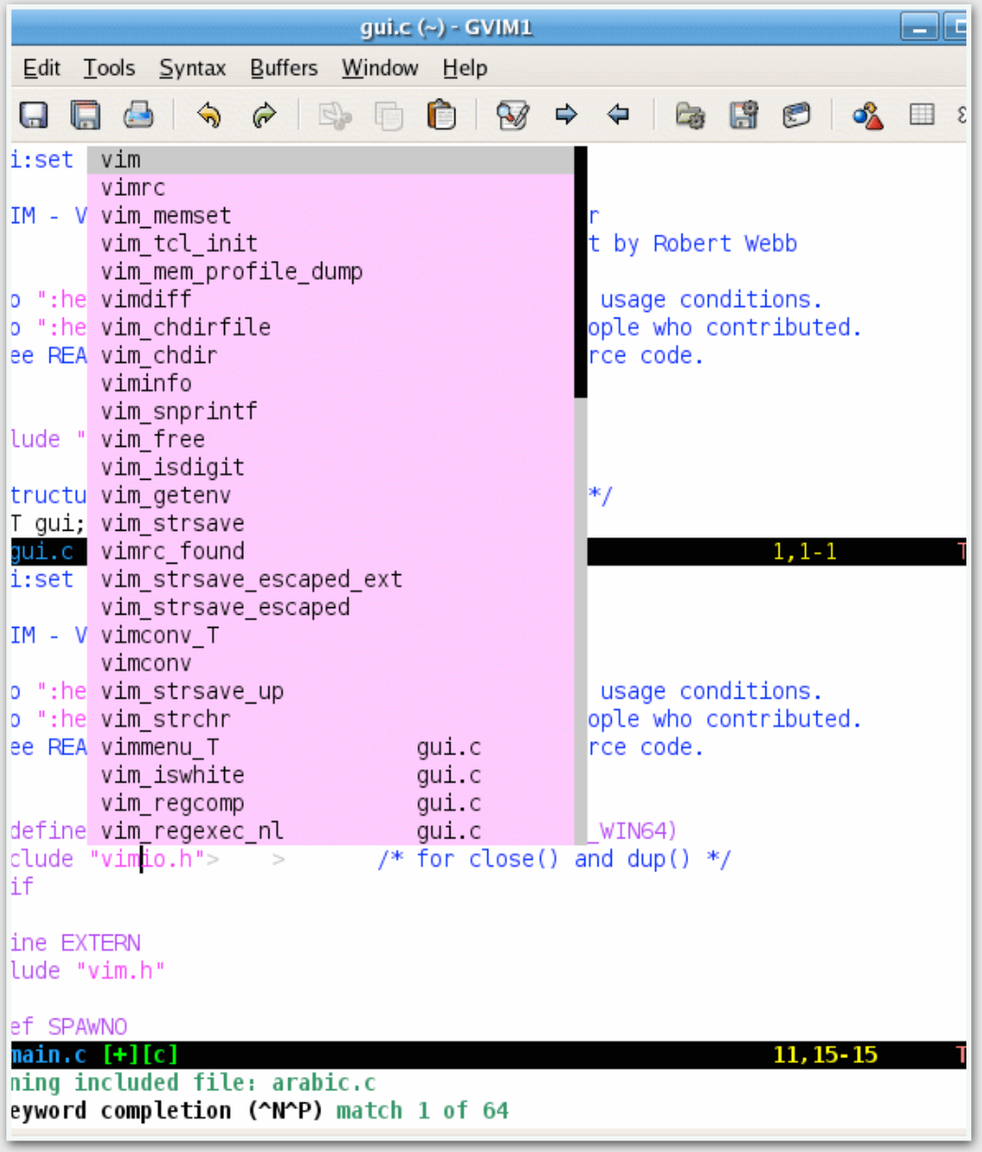
- Vim on Reddit:
  http://www.reddit.com/r/vim/

Why Vim

- Highly customisable
- Fits every programming language
- Strong OS integration
- Powerful scripting language

Hello Vim

- Run vim for in-terminal editing
- Use gvim to open a graphical editor window
- Download from: http://www.vim.org/download.php



Modal Editing Theory

- There are not enough keys in one keyboard
- Each key can have 4 meanings:
  - Normal
  - Editing
  - Visual
  - Command Line

Modal Editing Theory

Editing	Normal	Visual
h	h	h
Letter h	Move left	Select character to the left

Modal Editing Theory

- Lets you focus on current task:
  - Typing text
  - Searching and editing text

Normal Mode

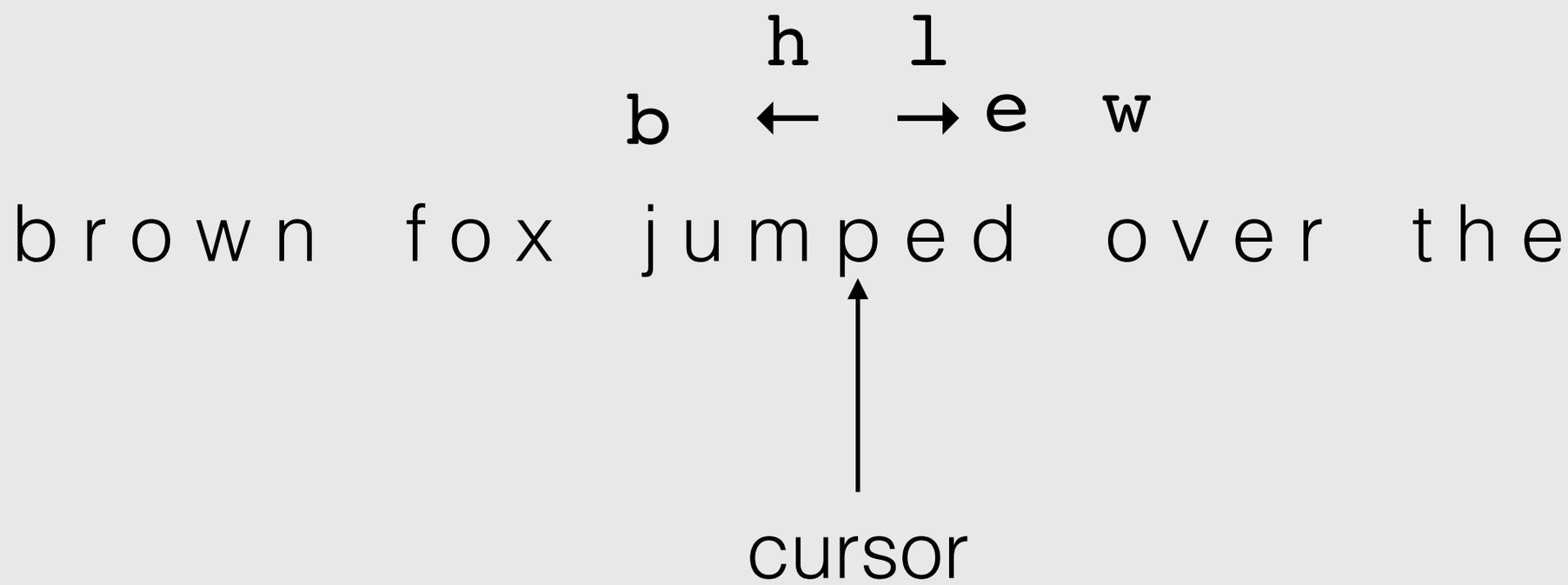
Normal Context

- Move around
- Search and read code
- Navigation in the buffer

Normal Commands

- In normal mode, every command takes the form: <count><command>[operates_on]
- Example:
  - d is the delete command
  - w is the word target
  - 4dw: delete 4 words

Motion Commands



Normal Commands: Delete

- d<motion> - delete
- D - delete until end of line
- dd - delete entire line
- x - delete character under cursor
- X - delete character to the left (backspace)

Normal Commands: Yanking

- y<motion> - yank
- Y - yank till end of line
- yy - yank entire line
- p - paste after cursor
- P - paste before cursor

Normal Mode Undo

- u - undo
- <C-r> (hold down ctrl and click r) - redo

Normal Mode Commands

- Try to figure out what the following do:
- gg, dgg
- G, dG, 7G
- f<letter>, F<letter>
- /<word>

Normal Mode Lab

- Copy /etc/passwd to your home folder
- Start vim on the new copy
- Delete the first line
- Move the next 4 lines to the end of the file
- Save and quit (use ZZ)

Insert Mode

Insert Context

- Typing characters adds text (like other editors)
- <Esc> back to normal mode
- Special commands are performed using modifier keys
- Vim users don’t stay here longer than we must

Normal -> Insert

- i - keep cursor position
- a - move cursor to the next character
- o - move cursor to the next line
- I - move cursor to beginning of line
- A - move cursor to end of line

Insert Mode Commands

- Try to figure out the following in insert mode:
- Ctrl + u
- Ctrl + y
- Ctrl + t
- Ctrl + n

Insert Mode Lab

- Start vim on existing file
- Delete first paragraph
- Write a new first paragraph telling 5 things you love about vim
- Save and Quit (from normal mode)

Command Line Mode

Command Line

- Vim has an internal command line for complex commands (not the unix command prompt)
- Use : to get there
- Use <Esc> to get back to normal mode

Command Line

- :q - close window
- :w filename - save as
- :wq - save and exit
- :e - open a file for editing
- :r filename - read file and paste its content into current buffer
- :r !cmd - run unix command and paste its output into current buffer

Command Line

- vim keeps its own current working directory. Can read and change using:
  :cd, :pwd
- !cmd - run a unix command
- !cmd % - run a unix command on current file

Command Line

- Try to figure out the following:

  :!ls
  :%!ls
  :pwd
  :e .

Command Line Lab

- Open the file /etc/passwd in vim
- Save the first 10 lines to another file, using a single command in the command line

Visual Mode

Visual Context

- Designed for selecting text in a visual way
- In visual mode, commands will operate on selected text
- Try selecting some text and press d, y, p, ~
- Use <Esc> to get back to normal mode

Visual Commands

- Enter visual mode using v, V, <C-v>
- v - enter visual mode selecting characters
- V - enter visual mode selecting lines
- <C-v> - enter visual mode selecting blocks

Vim help

- Use :help to get help on any topic
- Try the following:
  - :help del
  - :help edit
  - :help 42

Q & A

Working With Buffers

- Each window in vim displays a buffer
- Multiple buffers can be opened at the same time
- Window can switch active buffer

Buffer Commands

- :ls - show open buffers
- :b[id], :b name - move to buffer by id/name
- :bp, :bn - move to next/previous buffer
- :bd - close current buffer
- ^ - move to previous buffer

Demo

- Open a project with many files
- Search for buffers and move around

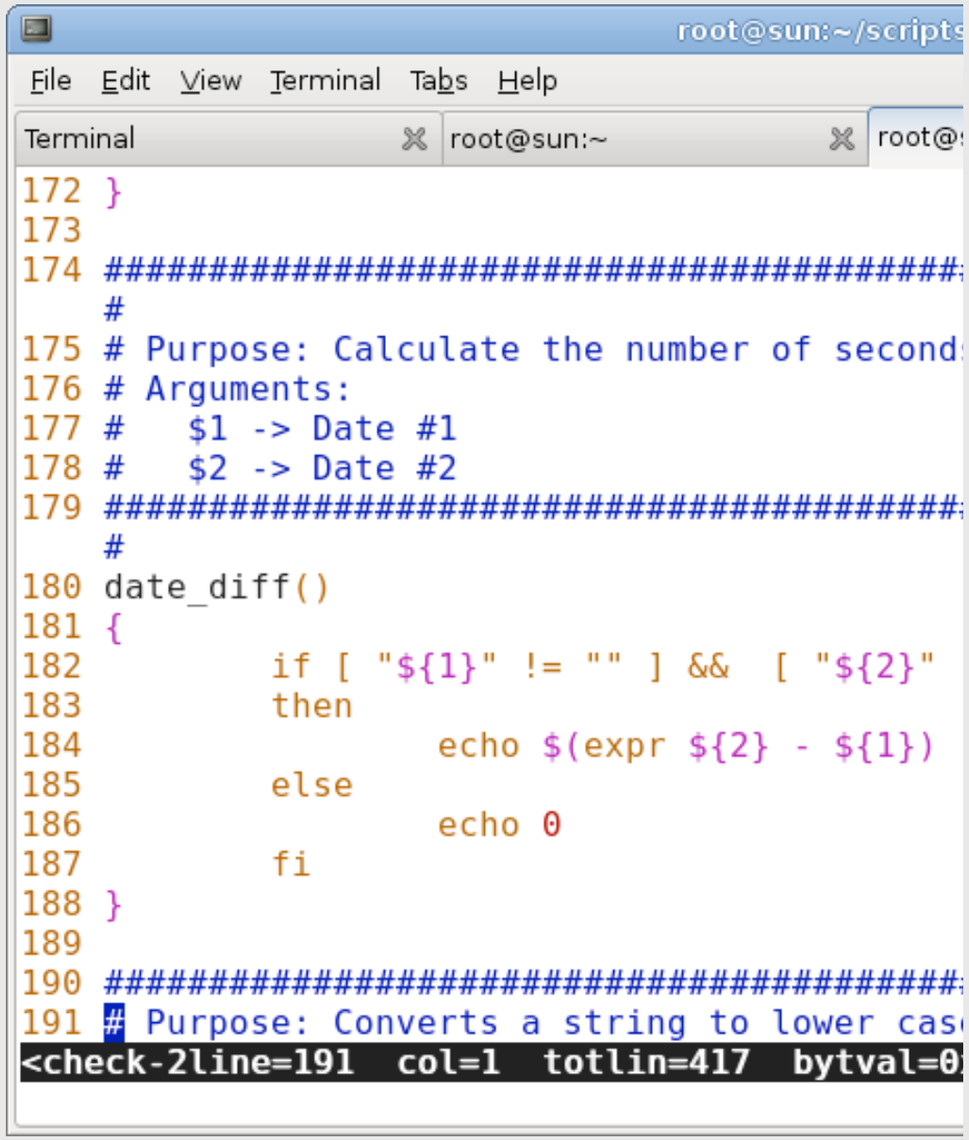
Build Your Own Vim

.vimrc file

- On vim startup, it looks for a .vimrc file in the user's home directory
- If such a file exists, every line is executed
- Next are some useful .vimrc commands

Line Numbers

- :set number turns on line numbers
- :set nonumber turns them off
- In vimrc, use without colon



Case insensitive search

- :set ic “ignore case when searching”
- Add to .vimrc without the colon

Key Bindings

- Vim lets us map any key sequence to any action
- map binds a key in all modes
- imap binds a key in insert mode
- cmap binds a key in command line (ex) mode
- nmap binds a key in normal mode
- vmap binds a key in visual mode

Key Bindings

- Bind C-s to the key sequence <Esc>, :w, enter, and then back to insert mode

- :imap <C-s> <Esc>:w<cr>a

Key Bindings ala Windows

- vmap <C-c> y
- imap <C-v> <Esc>p

Key Bindings ala Windows

- Open a new file dialog
  nmap <C-n> :browse confirm e<cr>

- Open a save as dialog
  map <C-S-s> :browse confirm saveas<cr>

A nice .vimrc starter

- Many people publish their .vimrc on github
- Find cool ones here: https://github.com/search?q=.vimrc

Q & A

Vim Plugins

- Provide extra functionality in vim
- Usually written in vimscript
- There’s a plugin for that

Where to find them

- http://vim-scripts.org/vim/scripts.html
- http://www.vim.org/scripts/index.php
- dedicated github repositories

Demo

Installing the NERDTree plugin

Recommended Plugins

- NERDTree
- Surround
- Align
- Command-T
- NERDCommenter

Directory Structure

- Contents of your ~/.vim/
  - `colors/` is where vim stores color schemes
  - `ftplugins/` are only used for a specific file type
  - `plugin/` for general purpose plugins
  - `syntax/` for syntax highlightning

A Full Vim Solution

- Configuration files + plugins = distribution
- Recommended distributions:
  - janus
  - spf13

Vim Summary

- A powerful text editor already installed on your unix box
- Designed for productivity

Online Vim Talks

- Conway at OSCON2013
  http://programming.oreilly.com/2013/10/more-instantly-better-vim.html

- Bram Moolenaar 7 habits
  http://www.youtube.com/watch?v=p6K4iIMloul

- Ben Orenstein: expert level vim
  http://www.youtube.com/watch?v=SkdrYWhh-8s

Q & A

Organising Files

Permissions, inodes and links

Unix Permission Model

- Every operation on a file or directory requires either:
  - Read permission
  - Write permission
  - Execute permission

Required Permissions

Action	Permission	On
Show file's contents	Read	file
Edit a file	Write	file
Execute a file (as script or binary)	Execute	File

Required Permissions

Action	Permission	On
Delete a file	Write	containing folder
Rename a file	Write	containing folder
List files	Execute + Read	containing folder

Checking Permissions

- Every file has 3 permission attributes matching the 3 types of users who access the file
- Owner’s permissions
- Group’s permissions
- Everyone else

Checking Permissions

Owner	Group	Other
R | W | -	R | - | -	R | - | X

Checking Permissions

Owner	Group	Other
R | W | -	R | - | -	R | - | X

User joe from group staff is trying to read the file

Checking Permissions

Owner	Group	Other
R | W | -	R | - | -	R | - | X

User joe from group staff is trying to read the file

Is joe the file's owner ?

Checking Permissions

Owner	Group	Other
R | W | -	R | - | -	R | - | X

User joe from group staff is trying to read the file

Is joe the file's owner ?

Does joe belong to the file's owning group ?

Listing Permissions

```
Permissions   Owner    Group

-rw-r--r--  1 ynonperek  staff  47440 20:33 29 אוק Makefile
-rw-r--r--  1 ynonperek  staff    369 20:33 29 אוק main.cpp
-rw-r--r--  1 ynonperek  staff   2684 20:33 29 אוק main.o
drwxr-xr-x  3 ynonperek  staff    102 20:33 29 אוק qtest.app
-rw-r--r--  1 ynonperek  staff    288 20:33 29 אוק qtest.pro
```

Setting Permissions

- chmod takes a permission descriptor, either as letters or numbers
- Examples:
  - u+r add read permission to user
  - g+w add write permission to group
  - o+x add execute permission to other
  - a-x remove execute permission from everyone

Setting Permissions

- Can you tell what happens to the file after each line ?

```
% touch main.c
% ls -l main.c
-rw-r--r--  1 ynonperek  staff  0 20:32 15 דצמ main.c

% chmod a+x main.c

% chmod g-rx main.c

% chmod o+w main.c

% chmod u+xg+xo-r main.c
```

Setting Permissions

- Setting the permissions in letters can be cumbersome
- Unix hackers are lazy
- So we have octals

Setting Permissions

4	2	1	4	2	1	4	2	1
r	w	x	r	w	x	r	w	x

Setting Permissions

4	2	1	4	2	1	4	2	1
r	-	x	r	-	-	r	-	-

Setting Permissions

4	2	1	4	2	1	4	2	1
r	-	x	r	-	-	r	-	-

5 4 4

Setting Permissions

4	2	1	4	2	1	4	2	1
r	-	x	r	-	-	r	-	-

5 4 4

chmod 544 file.txt

Permissions Worth Remembering

Numeric Value	Permission
755	`rwxr-xr-x`
644	`rw-r--r--`
600	`rw-------`

Pop Quiz

- What do each of the following permissions mean?
- 510
- 751
- 750
- 666
- 004

Q & A

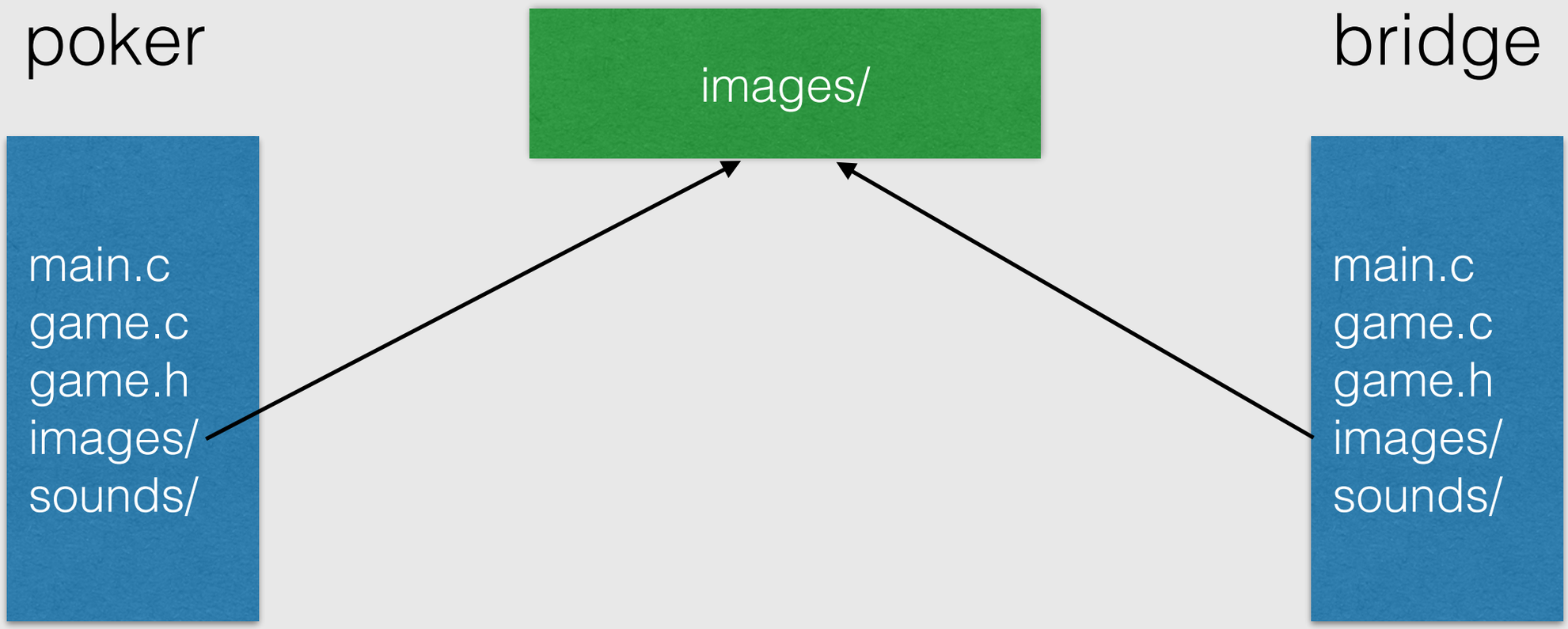
Lab: Permissions

Filesystem Links

File Links

- Links are special kind of files, which stand for other files
- Think of them as shortcuts
- Unix provides 2 kinds:
  - Hard links
  - Symbolic links

File Links


poker
images/
bridge
main.c
game.c
game.h
images/
sounds/
main.c
game.c
game.h
images/
sounds/

Symbolic Links

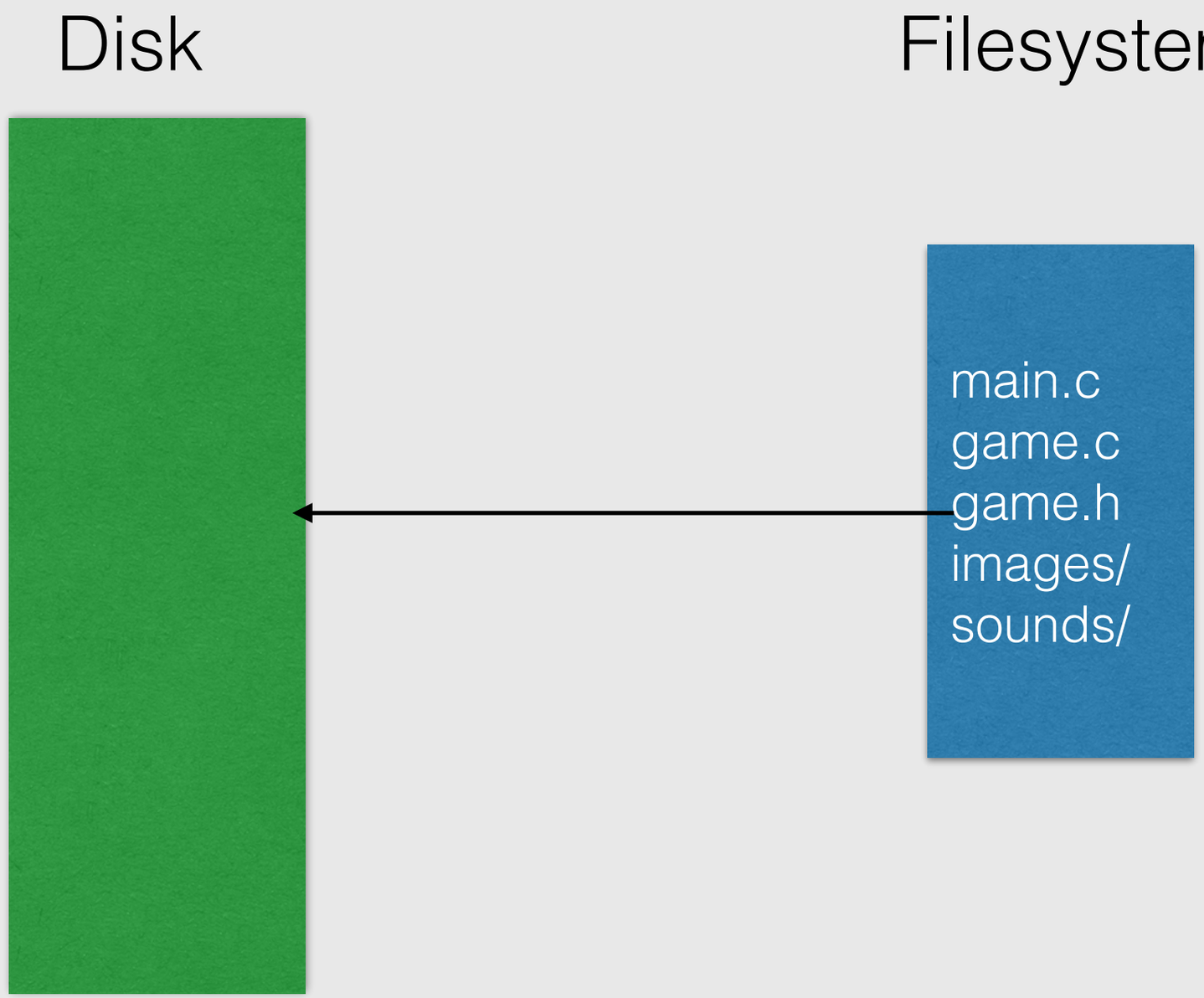
- A file has the name of another path written inside
- Demo

```
lrwxr-xr-x  1 ... passwd -> /etc/passwd
```

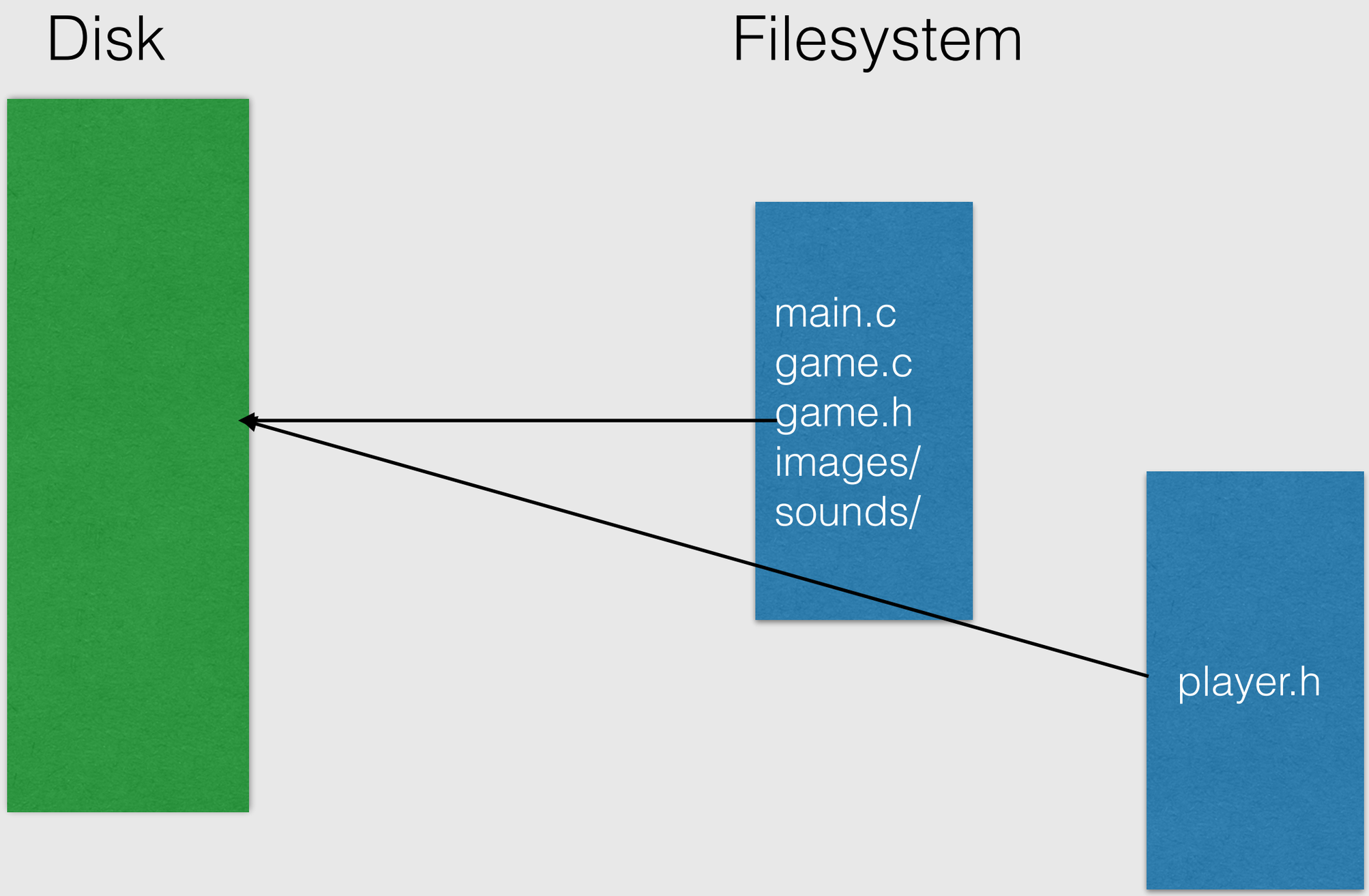
Symbolic Links

- Create with:
  - `ln -s <source> <destination>`

Hard Links



Hard Links



Hard Links

- Create with:
  - `ln <from> <to>`
- Can only link on the same file system

Hard vs. Symbolic Links

- Pop Quiz:
  - When a symbolic link is deleted, what happens to the other file ?
  - When a hard link is deleted, what happens to the other file ?

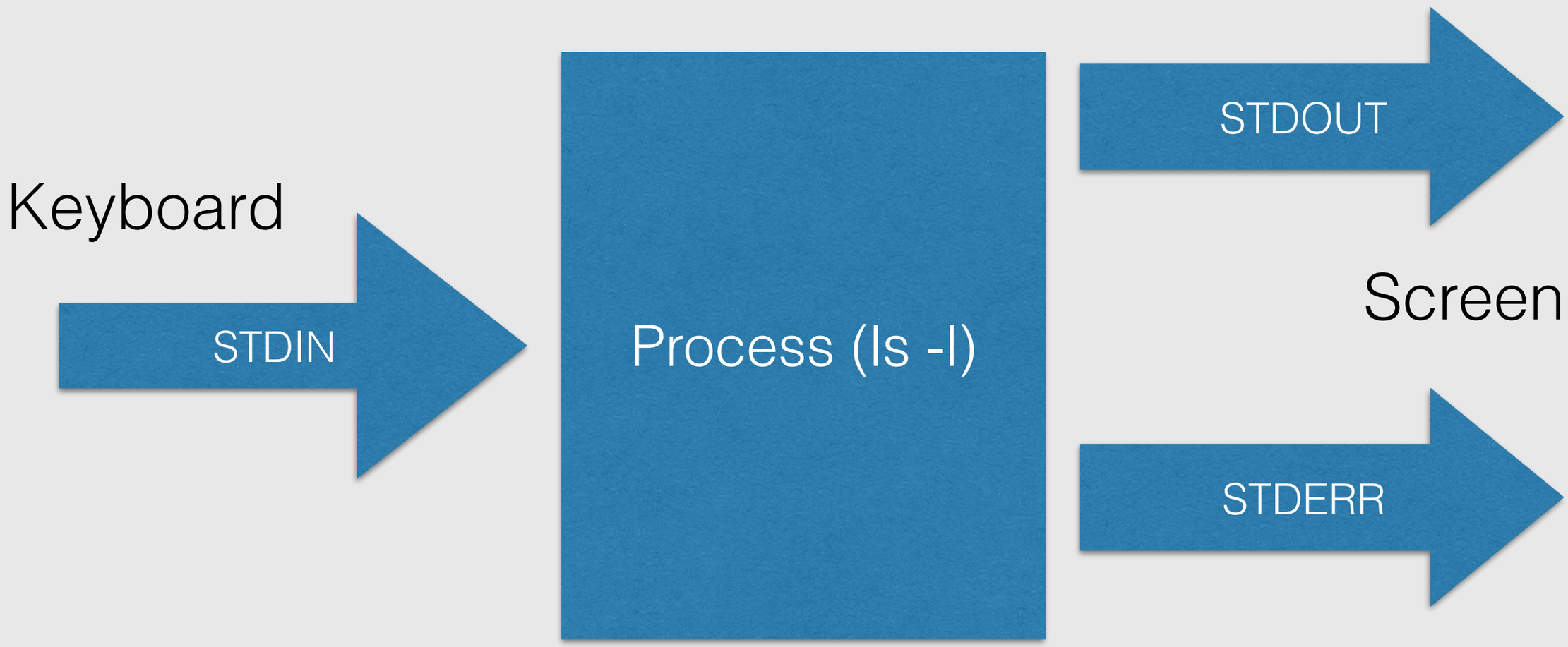
Lab: Links

I/O Redirections

Agenda

- Command Input and Output
- Pipes theory
- Common unix filters

I/O File Descriptors



I/O File Descriptors

- Each process has 3 descriptors
  - STDIN, STDOUT, STDERR
- By default they are connected to screen and keyboard
- You can tell the shell to redirect them

Redirecting Output

```
# prints to screen
% cowsay Hello

# prints to file
% cowsay Hello > cow.art

# append to file
% cowsay Hello >> cow.art
```

Redirecting Errors

```
% touch main.c
% ls main.c nosuchfile.txt
ls: nosuchfile.txt: No such file or directory
main.c

% ls main.c nosuchfile.txt > filelist
ls: nosuchfile.txt: No such file or directory
```

Redirecting Errors

- Errors are printed to STDERR
- Redirecting STDOUT (with >) has no effect

Redirecting Errors

```
# sends both STDOUT and STDERR to file
% ls main.c nosuchfile.txt >& alloutput

# sends STDOUT to file named ok
# and STDERR to a file named err
% ls main.c nosuchfile.txt 2> ok 1> err
```

Clobbering - Bash

```
% touch testfile
% ls > testfile
% set -o noclobber
% ls > testfile
testfile: File exists.

% ls >| testfile
% ls > testfile
testfile: File exists.

% set +o noclobber
% ls > testfile
```

Q & A

Pipes and Filters

Theory

- Instead of connecting STDOUT to a file, we can connect it as STDIN of another program

Theory

Longer Chains Are OK Too

```
 _______
<   10 >
 -------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

The Good

- No need to invent temporary file names
- Less typing -> more fun

More Pipes

```
% cat one.txt two.txt | sort

% ls | wc -l | cowsay

% date | cowsay
```

Using Pipes

- A program that takes input from STDIN and prints out to STDOU is called a filter
- Filters are used with pipes to solve bigger problems
- Examples:
  - head, tail, tee, cut, tr, sort, uniq, wc, cowsay, grep, awk, sed

Filters: tee

- Use tee to save a copy of the data

```
% ls | tee list | wc -l

% cat file1.txt file2.txt | tee files | wc -l

% ls | tee list | grep txt | wc -l
```

Get first / last lines

- Use head to get just the first few lines
- Use tail to get just the last few lines
- Use tail +n to get “from line n and onwards”

Head Demo

```
% ls -l | head -4

total 834504
-rw-r--r--     1 ynonperek  staff       1349 Dec 10 08:27 1
-rw-r--r--     1 ynonperek  staff       1348 Dec 10 08:27 2
drwxr-xr-x     3 ynonperek  staff        102 Sep 24 16:13 Apps
```

Tail

- Use tail -f to follow a file as it is written
- Demo

cut

- Some unix data is column based
- cut lets you take just parts of it

cut

```
% who

ynonperek console  Feb 22 10:50
ynonperek ttys000  Mar  1 17:33
ynonperek ttys001  Mar  2 21:59
ynonperek ttys002  Mar  2 22:01
```

cut

```
% who | cut -f2 -d' '

console
ttys000
ttys001
ttys002
```

cut

- -f specifies field number
- -d specifies delimiter
- -c cuts characters
- Example:
  - `cut -c1-4,8`

tr

- Converts one character set to another
- Character set is simply a collection of characters (such as a-z, A-Z, 0-9, etc.)

tr

```
% tr 'a-z' 'A-Z' story.txt

% tr '0-9' '.' phones.txt

% tr 'a-zA-Z' 'A-Za-z' story.txt

% tr -d '0-9'

% tr -s ' '
```

sort

- Use sort to sort some input
- By default sort ascending as text strings
- Use -n for numerical sort
- Use -f to ignore case
- Use -k to pass a sort key column
- Use -u to sort only unique lines

sort gotcha

```
# Don't do this (won't work):
% sort data > data

# Remember the -o (this works):
% sort data -o data
```

grep

- Use grep to get just a few lines from entire output
- Use -v for inverse
- Use -x for full line match
- Use -ilR for “find in files”

grep

```
% cat /etc/passwd | grep root

% ls -l | grep ynonperek

% /sbin/ifconfig |grep inet | grep -v inet6 | grep -v 127.0.0.1
```

Q & A

Lab: Redirections and Pipes

Unix Power Tools

- diff
- find
- Regular Expressions

diff

- diff <from-file> <to-file>
- generate diff list between two files
- later use patch to update original file

diff

```
# Create a diff file
% diff main.cpp main_new.cpp > main.diff

# view and edit diffs
% vim main.diff

# apply patch to main.cpp
% patch main.cpp main.diff
```

diff

- Use -b to ignore change in white spaces
- Use -B to ignore change in blank lines
- Use -i to ignore changes in case
- Use -q to report only if files differ
- man diff for more info

find

- Finding files by type
- Finding files by owner
- Finding files by name
- Finding files by size
- Finding files by modification date

find by type

```
# find directories
% find . -type d

# find normal files
% find . -type f

# find symbolic links
% find . -type l
```

find by owner

```
# find by user name
% find  . -user joe

# find by user id
% find . -user 501
```

find by name

```
# find all files named main.c
% find . -name main.c

# find all files ending with .o
% find . -name '*.o'

# find all directories named src
% find . -name src -type d
```

find by size

```
# find all files larger than 2k
% find . -size +2k

# find all files smaller than 2mb
% find . -size -2M

# find all files larger than 2k but smaller than 5k
% find . -size +2k -size -5k
```

Lab: Find

Regular Expressions

Finding a pattern in the text

The Regexp Story

- Started in Mathematics
- 1968 Entered the Unix world through Ken Thompson's qed
- 1984 Standardized by Henry Spencer's implementation

Regular Expressions & Unix

- Many UNIX tools take regular expressions
- grep/egrep filters its input based on regular expressions
- more/less/most search uses regular expressions
- vi/vim search and replace use regular expressions

Regular Expressions Today

- Used by all programming languages, including:
  - Php, Tcl
  - Python, Perl, Ruby
  - JavaScript, ActionScript,
  - C#, Java
  - C, Objective C, C++
  - And More

Rule #1

A Simple character matches itself

Examples

Command	Meaning
egrep foo	Display only input lines that include the word ‘foo’
egrep unix	Display only input lines that include the word ‘unix’

Rule #2

A character class matches a single character from the class

Character Classes

a-z	0-9	0-9
a	2	2
z	7	4
b	2	1
c	3	3

Character Class Syntax

- A class is denoted by [...]
- Can use any character sequence inside the squares
  [012], [abc], [aAbBcZ]
- Can use ranges inside the squares
  [0-9], [a-z], [a-zA-Z], [0-9ab]
- Can use not
  [^abc], [^0-9]

Examples

Command	Meaning
egrep ‘[0-9][0-9]’	Display only input lines that include at least two digits
egrep ‘[Uu][Nn][Ii][Xx]’	Display only input lines that include the word ‘unix’ in any casing

Pop Quiz: Which Ones Match ?

hello [ux][012]

hello world
hello unix
hello u2
hello x10
HELLO U2

Answer

hello [ux][012]

hello world

hello uni0

hello u2 ✔

hello x10 ✔

HELLO U2

Rule #3

A quantifier denotes how many times a letter will match

Quantifiers

- * means match zero or more times - {0,}
- + means match one or more times - {1,}
- ? means match zero or one time - {0,1}
- {n,m} means match at least n but no more than m times
- {n} means match exactly n times

Which of these match ?

[0-9]{2}-?[0-9]{7}

08-9112232

421121212

054-2201121

Phone: 03-9112121

Bond 007

Which of these match ?

[[:digit:]]{2}-?[[:digit:]]{7}

08-9112232 ✓

421121212 ✓

054-2201121 ✓

Phone: 03-9112121 ✓

Bond 007

Which of these match ?

(http://)?w{3}\.[a-z]+\.com

www.google.com

www.ynet.co.il

http://mail.google.com

http://www.home.com

http://www.tel-aviv.com

Which of these match ?

(http://)?w{3}\.[a-z]+\.com

www.google.com ✓

www.ynet.co.il

http://mail.google.com

http://www.home.com ✓

http://www.tel-aviv.com

Rule #4

An assertion will match on a condition, not capturing input characters

Assertions

- ^ matches the beginning of a line
- $ matches the end of a line

Which of these match ?

^d

drwxr-xr-x dive

-rwxr-xr-x dive

lrwxr-xr-x dive

drwxr-xr-x /home

-rwxr-xr-x /etc/passwd

Which of these match ?

^d

drwxr-xr-x dive ✔

-rwxr-xr-x dive

lrwxr-xr-x dive

drwxr-xr-x /home ✔

-rwxr-xr-x /etc/passwd

Regular Expressions Variants

- Old style regexps didn't have a + or {}. Therefore, in grep, these have to be backslashed
- Use egrep when possible for cleaner syntax

Q & A

Lab: Regexp

Unix Environment

Agenda

- History
- Aliases
- Environment variables
- Startup files

CSH History

- Use history to show all past commands
- Use !n to execute a historic command
- Use !* to get all arguments from last command

Shell Aliases

Aliases

- An alias is a shorthand for a longer command line
- Aliases are saved in shell's memory, and deleted when shell exits

Aliases

- Define an alias with:
  - `alias <short>=<long>`
- Remove an alias with:
  - `unalias <short>`

Aliases

- Create new commands
- Modify behaviour of existing commands

Modifying Behaviour

- Can you tell what the followings do ?

```
alias ll="ls -l"

alias cp="cp -i"

alias rm="rm -i"

alias cf="ls -l | wc -l"
```

Modifying Behaviour

- You can always perform non-aliased behaviour by quoting
- Examples:
  - `\ls`
  - `\rm file.txt`

Adding Commands

- Can you tell what the followings do ?

```
alias c=clear

alias cd..="cd .."
```

Functions

- Aliases don’t take arguments, functions do

```
mycd() { mkdir "$1"; cd "$1"; }

hi() { echo "************ $* ************"; }
```

Environment Variables

- Unix processes take info from 2 sources:
  - Command line options
  - Environment variables
- Use environment variables for modifications you need in every run

Environment Variables

- We use environment variables to:
  - let man know which pager to use
  - let our mail app know which editor to use
  - let gcc know where common header files are

Environment How

- Use setenv to set an environment variable
  - `setenv PAGER less`
  - `setenv USERNAME ynonp`
  - `setenv http_proxy proxy-iil.intel.com:911`

Environment How

- Use unsetenv to remove an environment variable
- Use env to see a list of all environment variables

Special Environment Variables

Variable	Meaning
PATH	Search path for executables
USER	Current user
HOME	Home directory of current user
EDITOR	Favorite text editor

Startup Files

- All shell customisation is forgotten when shell exits
- Startup files are executed first thing when shell starts
- We can use them to get permanent customisations

Startup Files

- CSH has 4 startup files

File	Role
/etc/csh.login	system login
~/.login	user specific login
/etc/csh.cshrc	system rc file
~/.cshrc	use specific rc file

Other Startup Files

- .aliases is often used to store aliases
- You need to explicitly include it from one of the “normal” startup files

Lab: Aliases and Startup Files

Unix Process Management

- Process theory
- Unix daemons
- Watch running processes
- Run task in background
- Control background jobs
- IPC using signals

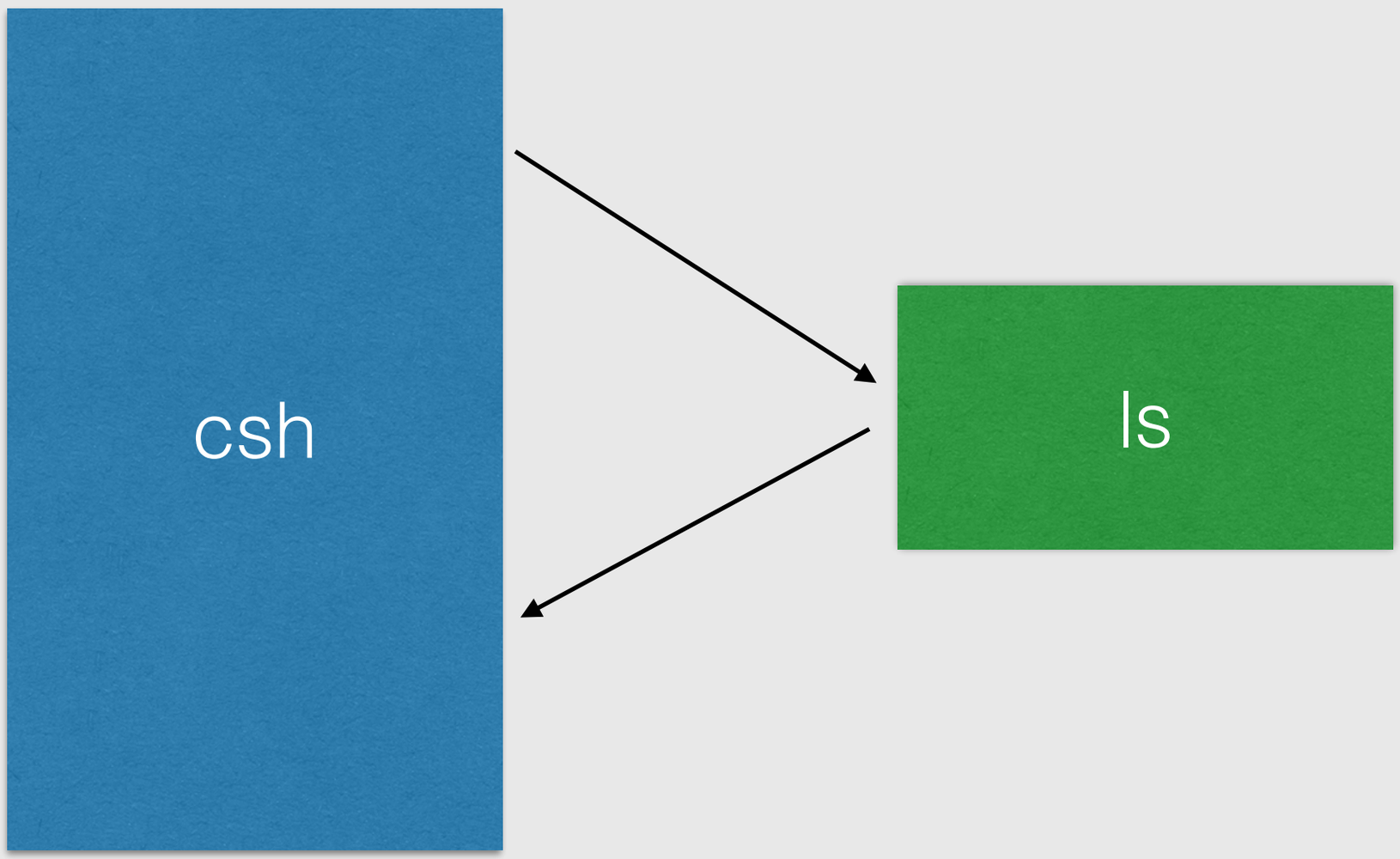
What’s a process

- A process is an instance of a computer program being executed
- A process has:
  - An executable program
  - Memory
  - Processor state

Where Processes Come From

- Every time your shell executes an executable, a new process is born
- When the process ends, you'll be back in the shell

Where Processes Come From



Other Processes

- Unix system processes are called deamons (because they work in the background)
- Examples:
  - Web server, DB server, ftp server, atd

Common Daemons

- Use atd to schedule tasks at specified times

```
# Schedule a new task at 10:00 tomorrow
at 10:00 tomorrow

# Schedule a new task on Jan 1st
at 'Jan 1 2014'

# See a list of all scheduled tasks
atq
4 Wed Jan  1 00:11:00 2014

# Remove task #4
atrm 4
```

Common Daemons

- Use crontab to schedule a repeating task
- Edit tasks with:
  - `crontab -e`
- View tasks with:
  - `crontab -l`

Common Daemons

```
#####################
# Cron fields
#   minute        0-59
#   hour          0-23
#   day of month 1-31
#   month         1-12
#   day of week  0-7 (0 or 7 is Sun)
#

# Backup the server daily
0 0 * * * /bin/backup.sh

# Backup the database every hour
0 * * * * /bin/backup_db.sh

# Send email once a week
0 0 * * 4 /bin/send_report.sh

# Check data every 5 minutes
0-59/5 * * * * /bin/check_data.sh
```

Cron Quiz

- When will the following commands run?

```
0,20,40 0-23/2 * * * touch alive

0 0 1 1 * /bin/greet.sh

13 13 13 * 5 /bin/play.sh
```

Running Processes

- Use `ps` to get a list of all running processes
- Use `top` to see each process resources

Background Execution

- Run a command in the background by adding an & to the end of the command line
- Each background process is called a job
- Use jobs to list all running jobs

Background Execution

```
# returns to shell after 10 seconds
% sleep 10

# returns immediately
% sleep 10 &
```

Job Control

- Press Ctrl+Z to put a running process "on hold", and go back to the shell
- Use jobs to see all jobs (frozen and background processes)
- Use `fg` to return a job to foreground (active process)
- Use `bg` to send a frozen task to work in the background
- Both `fg` and `bg` can take job number as % sign

Signals

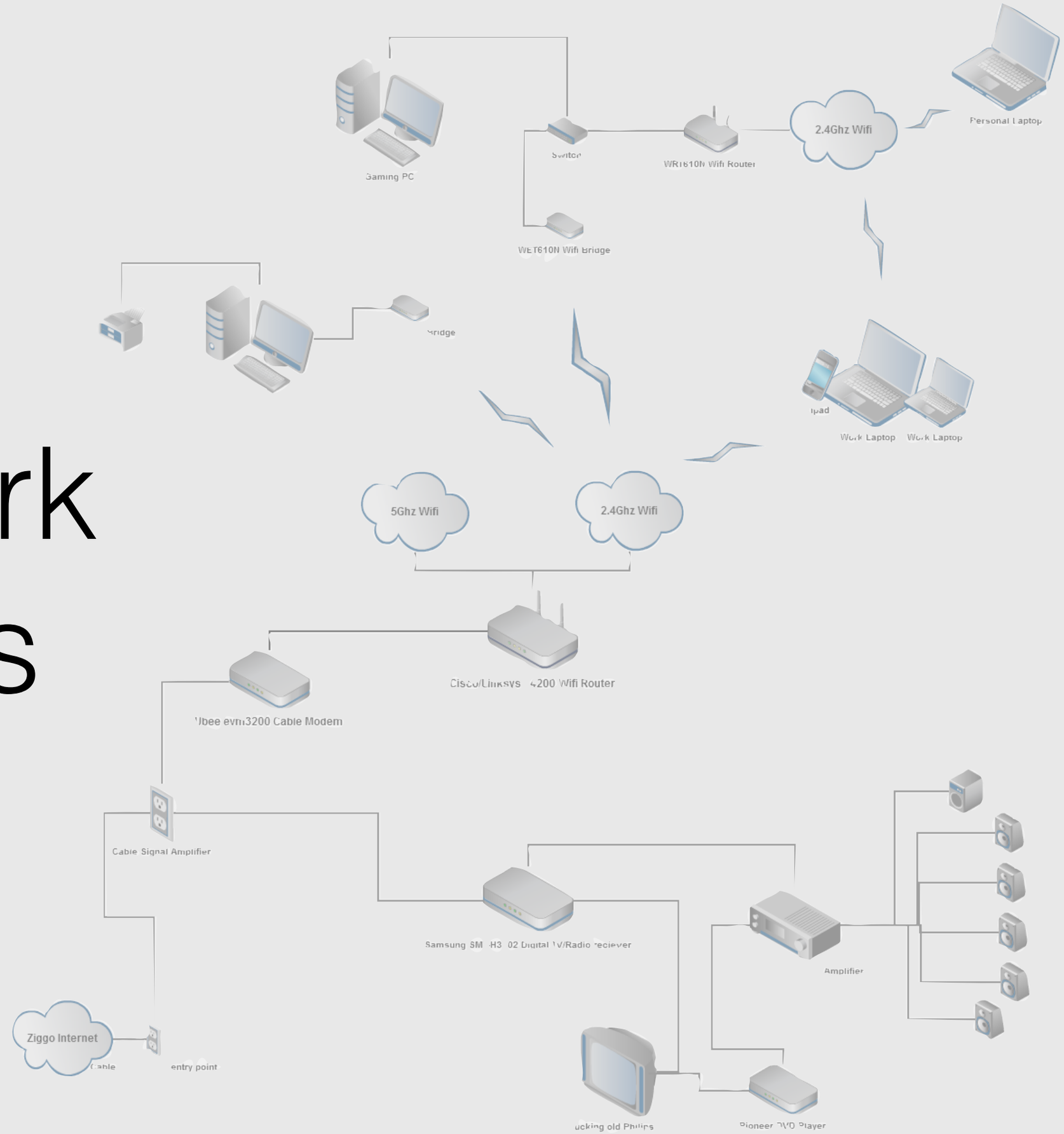
- The kill command sends a signal to another process

```
kill -l

 1) SIGHUP    2) SIGINT      3) SIGQUIT     4) SIGILL
 5) SIGTRAP   6) SIGABRT     7) SIGEMT      8) SIGFPE
 9) SIGKILL  10) SIGBUS     11) SIGSEGV    12) SIGSYS
13) SIGPIPE  14) SIGALRM    15) SIGTERM    16) SIGURG
17) SIGSTOP  18) SIGTSTP    19) SIGCONT    20) SIGCHLD
21) SIGTTIN  22) SIGTTOU    23) SIGIO      24) SIGXCPU
25) SIGXFSZ  26) SIGVTALRM  27) SIGPROF    28) SIGWINCH
29) SIGINFO  30) SIGUSR1    31) SIGUSR2
```

Q & A

Lab: Processes and Jobs

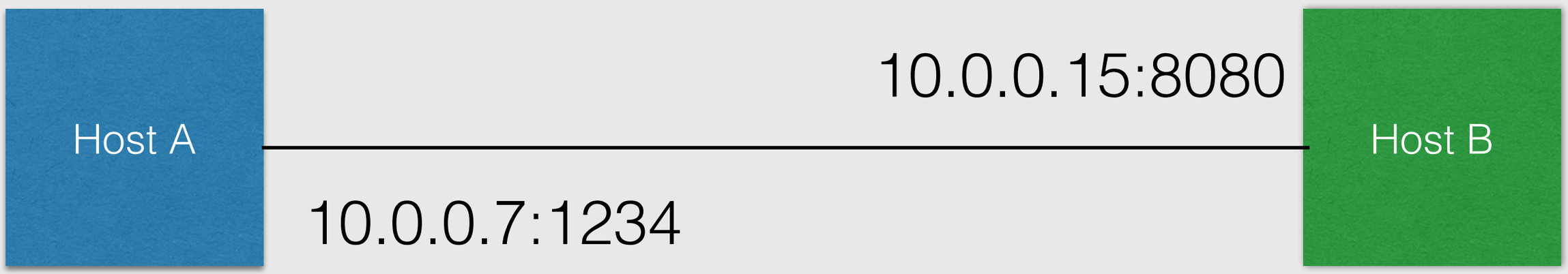
Network Utilities



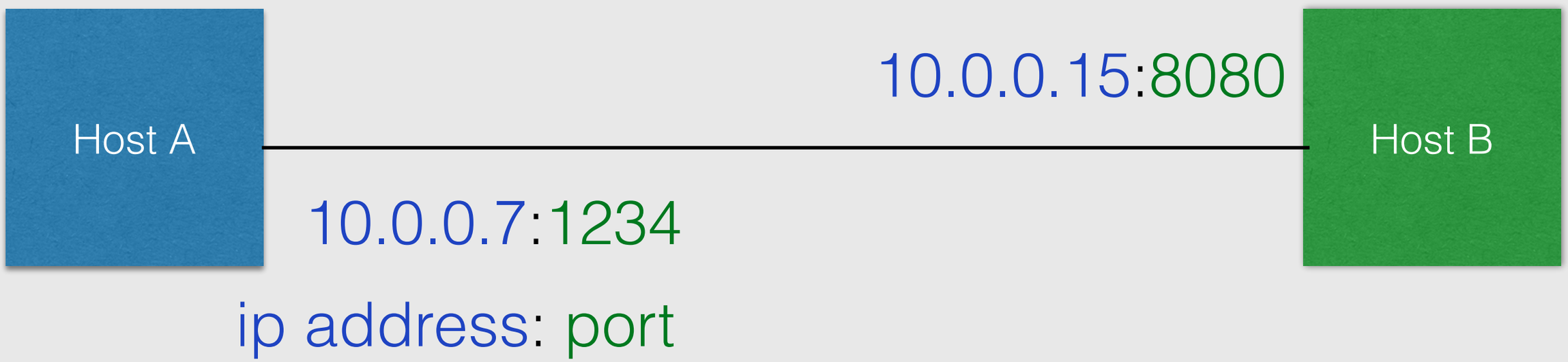
Scope

- We'll talk about:
  - TCP/IP
  - Network Protocols: HTTP, DNS, SMB, ICMP

TCP Socket



TCP Socket



TCP Socket

- A “channel” between hosts
- Identified by IP:Port <-> IP:Port
- Each host can write and read from the channel

TCP Socket Demo

Host A

```
nc -l -p 9898
```

Host B

```
nc 176.58.103.66 9898
```

TCP Sockets Workflow

- Host A listens for incoming connections
- Host B connects
- Usually, A spawns a new process to “talk” to B, and gets back to listening

Sockets Status

- Use `netstat` to get a list of all sockets on a host
- Use `netstat -p` to also get program names

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:49418         localhost:8009          ESTABLISHED
tcp        0      0 localhost:27017         localhost:45312         ESTABLISHED
tcp        0    208 li453-66.members.li:ssh 93-172-179-129.bb:59904 ESTABLISHED
tcp        1      0 localhost:54205         localhost:5011          CLOSE_WAIT
tcp        0      0 localhost:39168         localhost:27017         ESTABLISHED
tcp        0      0 localhost:mysql         localhost:46728         ESTABLISHED
tcp        0      0 localhost:27017         localhost:45340         ESTABLISHED
tcp        0      0 localhost:27017         localhost:39172         ESTABLISHED
tcp        0      0 localhost:27017         localhost:39168         ESTABLISHED
tcp        0      0 localhost:39171         localhost:27017         TIME_WAIT
tcp        0      0 localhost:48903         localhost:8009          ESTABLISHED
```

Domain Names

- Remembering IPs is hard and unfriendly
- Domain name is a name connected to an IP
- A domain name system translates from domain name to an IP

Domain Names

Host A
IP Address:
192.117.122.148

whatsup.org.il

Domain Names

- Use `dig` to get the domain details for a given domain name
- File `/etc/resolve.conf` keeps the name of your current domain name server
- File `/etc/hosts` keeps a list of local names

Q & A

Searching for hosts

- `ping` checks if a host responds
- `traceroute` shows the route to another host
- `nmap` runs a port scan on a host

Using Network

- chats
- file transfer
- remote shell
- web access

Chatting

- `write user` sends a short message to another user
- `talk user@other.machine` starts a conversation
- IRC is a really simple text based chat protocol. Clients: BitchX, iirc

File transfers

- Use `smbclient -L host` to get a list of windows shares on a host
- Use `smbclient //host/share` to start sending or receiving files from host

Remote Shell

- Use ssh `user@host` to open a secure shell on a remote host
- Use ssh-keygen to generate personal SSH keys
- Put the public key on the remote host's ~/.ssh/authorized_keys to get password-less access

Web Access

- Use `lynx http://www.google.com` to open google in a textual browser
- Use `curl http://en.wikipedia.org/wiki/Unix` to get Unix's page from wikipedia (as HTML text)

Web Access

- In most organisations, you'll need to set an HTTP proxy to access external world
- Use the environment variable http_proxy to set one

Q & A

Thanks For Listening

- Ynon Perek
- ynon@ynonperek.com
- http://ynonperek.com
- Lab Solutions: https://gist.github.com/ynonp/8542719